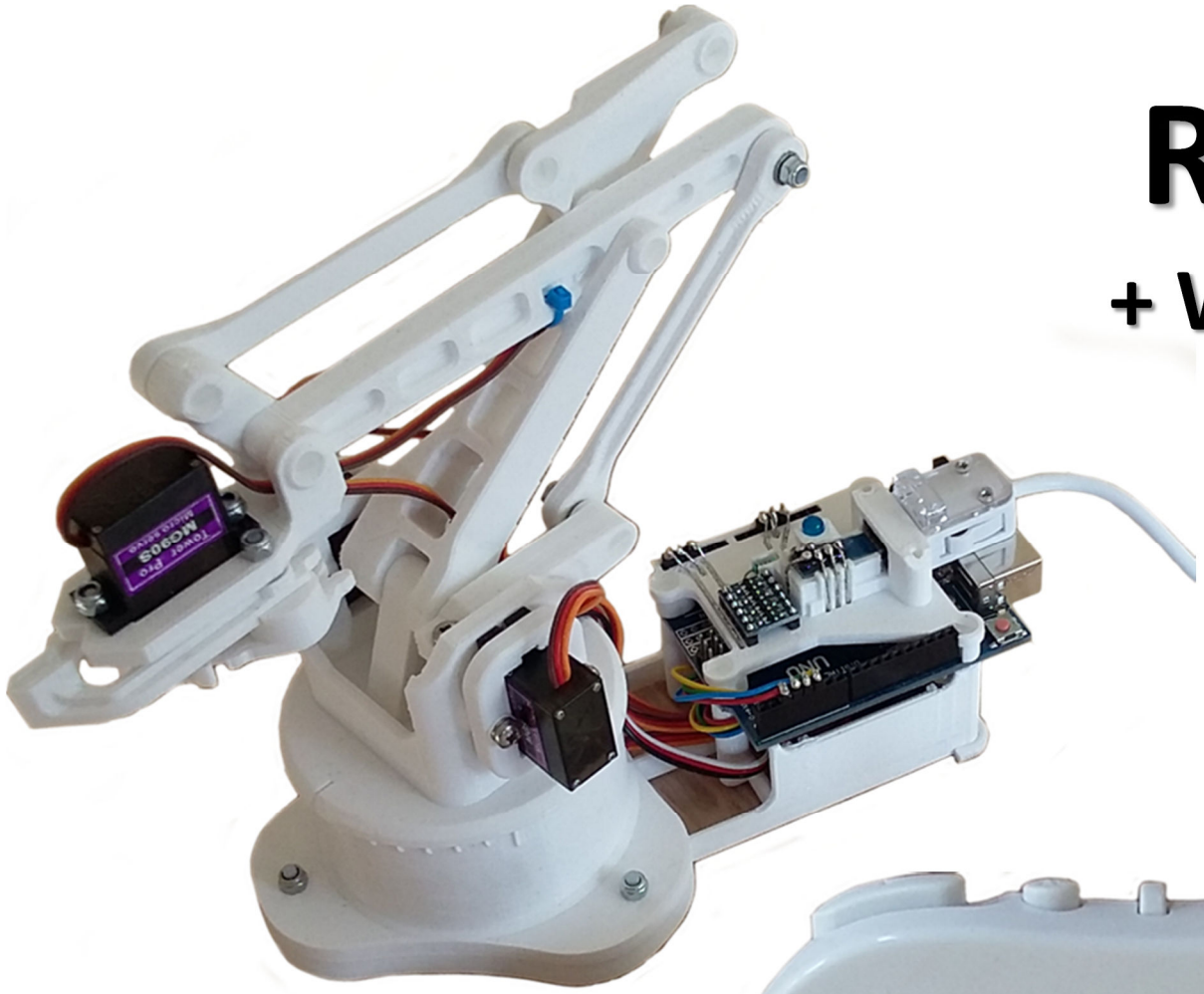
Reach Robot

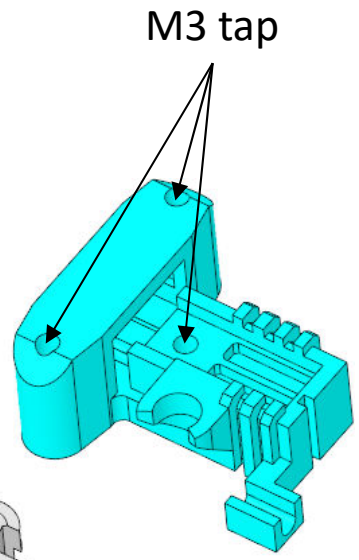
+ Wii Classic Controller



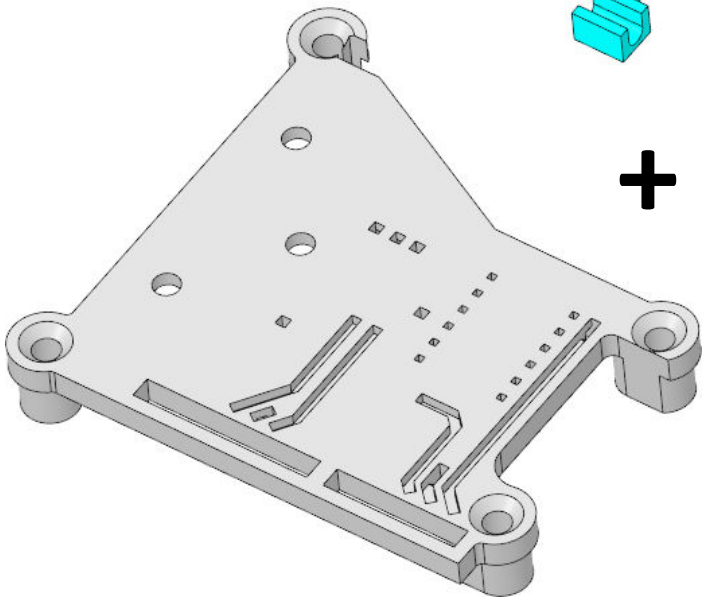
Includes:

- 3-D Model STL files
- Circuit diagram
- Arduino code

3-D Models

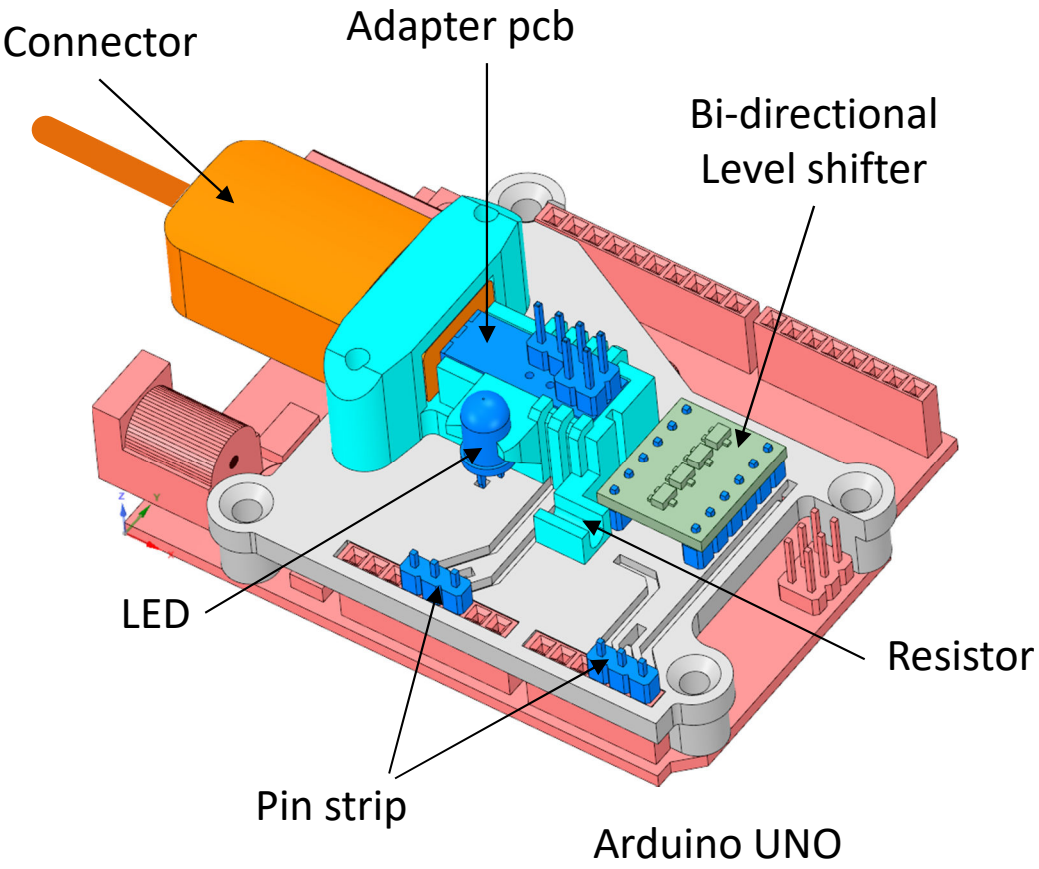


Upper Mount 01.stl



Mounting Plate 01.stl

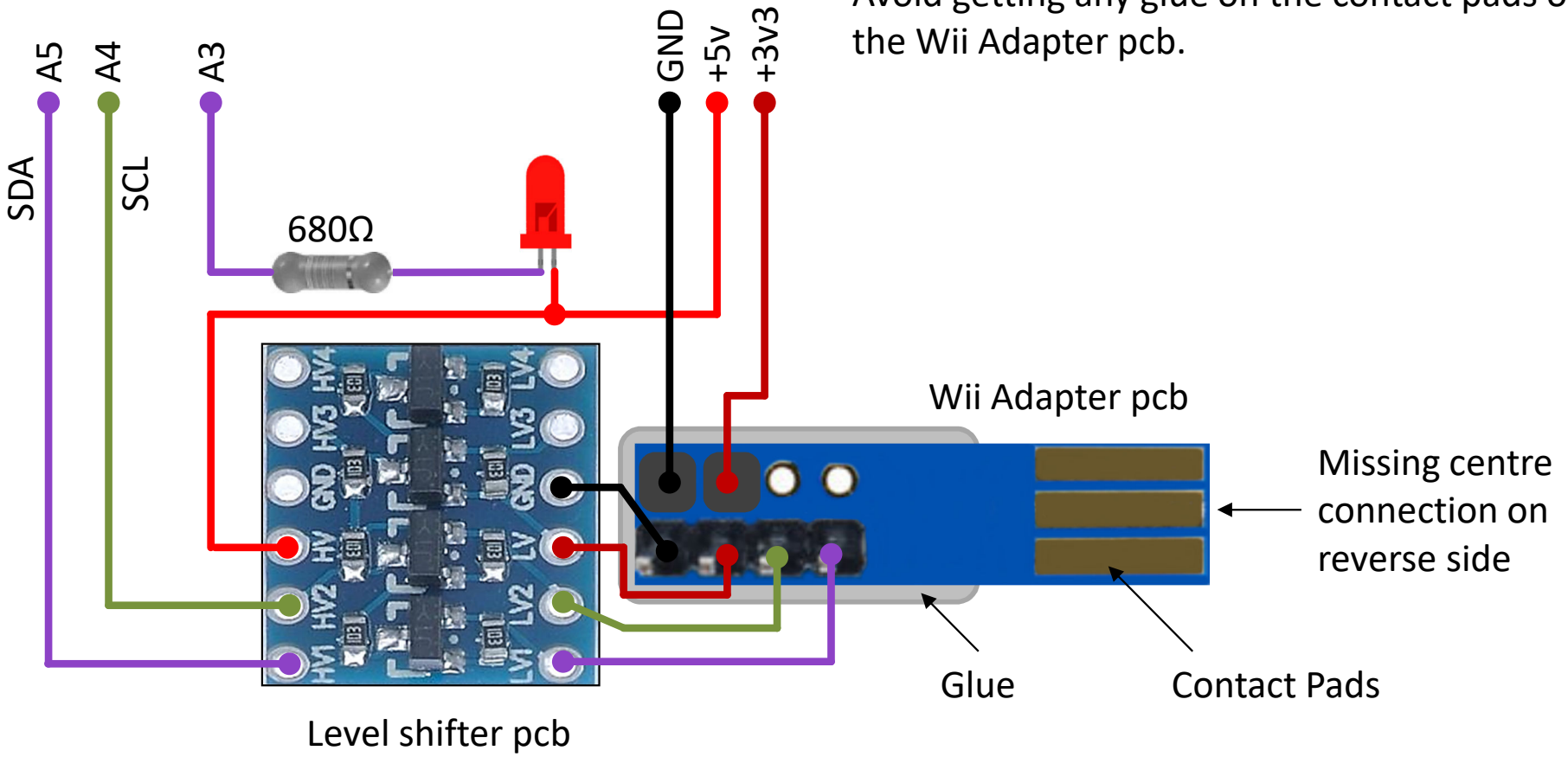
There are two 3-D parts to this design. A base Mounting Plate which mounts onto the Arduino UNO and routes the wiring between components. Plus an Upper Mount which supports the Wii adapter pcb, guides and retains the Wii connector plug and supports the LED components. They are held together using 3 x M3 countersunk nylon screws. You need to tap the holes in the Upper Mount with an M3 tap.



Circuit Diagram

The Wii Classic Controller interface uses standard I2C bus communications, which are present on the Arduino UNO on analog pins A4 and A5. To get this to work correctly you must use a bi-directional level shifter. I used half of a quad circuit you can easily acquire from eBay. The LED and associated resistor are optional and driven from code, which illuminates the LED whenever the presence of the Wii controller is detected on the I2C bus. The Wii adapter pcb was glued into the Upper Mount using quick-set epoxy glue prior to commencing the wiring.

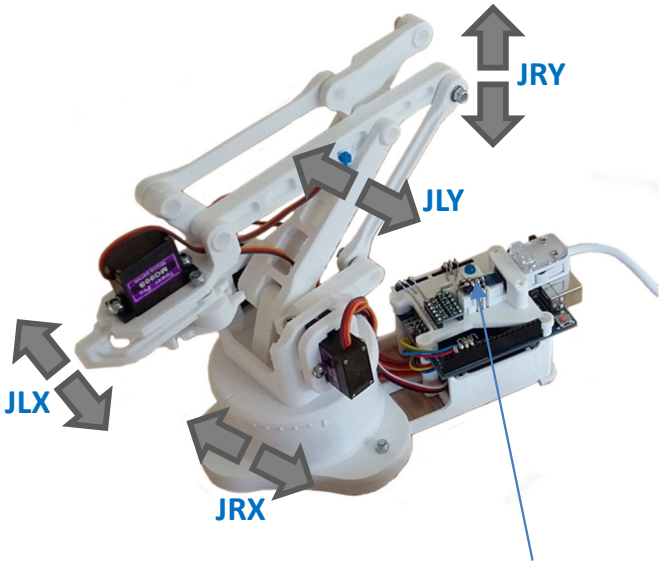
Avoid getting any glue on the contact pads of the Wii Adapter pcb.



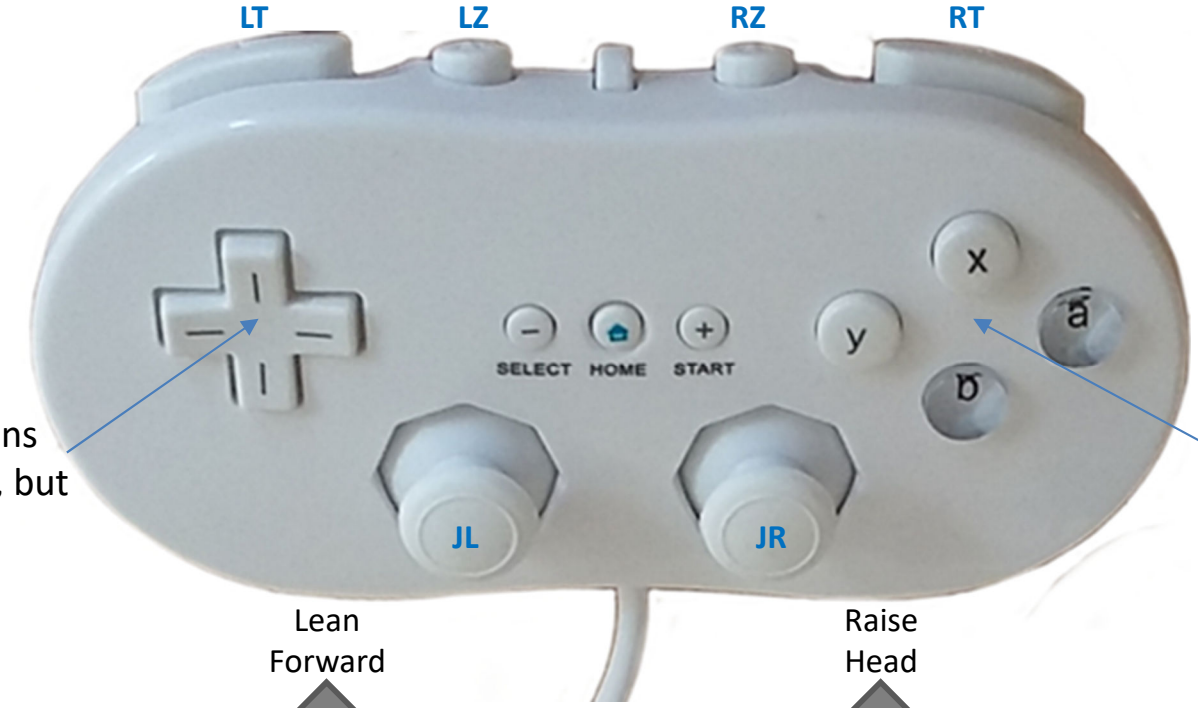
Code Functions

The sample code provided gives you the following features and functions:

HOME	- move to the 'HOME' position
SELECT	- move to the 'RESET' position
START + LZ + RZ	- perform 'Move' sequence
RT	- holding reduces joystick sensitivity by half
RZ	- holding reduces joystick sensitivity by quarter

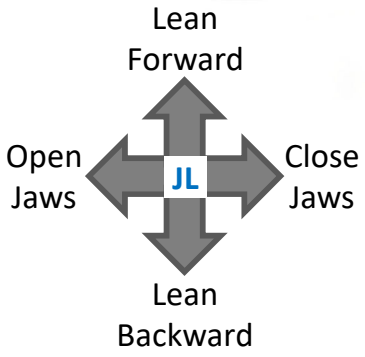


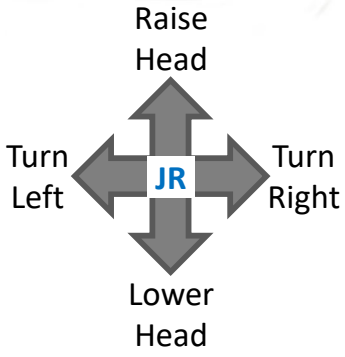
LED illuminates when the Wii Classic Controller is both connected and detected on the I2C bus. Otherwise it will be OFF.



Digital joystick buttons can be read by code, but are not used here.

These buttons can be read by code, but are not used here.





C++ Code

Sample code is provided in the Wii_Classic_Reach_Robot_00 folder, which you can study, use and modify as you see fit. Most of the code is extracted from library functions available on the internet.

These definitions are before void setup():

```
// define Wii specific items
# define WiiLED A3     // LED used to indicate Wii plugged in
byte WiiCnt;           // received byte counter
uint8_t WiiData[6];    // array to store Wii output
byte WiiError = true; // used to detect transmission errors
int WiiLX,WiiLY;      // left joystick values
byte wiiPhase = 0; // used to perform wii functions at 50 Hz
int WiiRX,WiiRY;      // right joystick values
```

This code defines the LED pin in void setup():

```
pinMode(WiiLED, OUTPUT);  // set output pin for WiiLED
digitalWrite(WiiLED,HIGH);   // turn OFF WiiLED
```

The code in the main loop starts with:

```
  if (WiiError == 0) {
    // Wii attached and responding so scan it
    digitalWrite(WiiLED,LOW);   // turn ON WiiLED
    wiiPhase--;
    if (wiiPhase < 1) {
      // in wiiPhase 0 every 20ms
      wiiPhase = 2; // reset phase
```

The Functions tab contains these Wii functions:

```
void wiiClearData() {
  // sets all data bits to 1's

void wiiData() {
  // send Wii reported data to the serial port

void wiiInitialise() {
  // called to check for Wii presence and initialise it if it is

int WiiLeftStickX() {
  // returns an LX value 0/32/63

int WiiLeftStickY() {
  // returns an LY value 0/32/63

boolean WiiPressedRowBit(byte row, byte bit) {
/* Read a bit to test for button presses. Call directly using the following
  values  (row,bit):

int WiiRightStickX() {
  // returns a RX value 0/16/31

int WiiRightStickY() {
  // returns a RY value 0/16/31

void WiiUpdate() {
  // reads 6 bytes of data from the controller over I2C

void WiiWriteZero() {
  // writes one zero byte to the Wii to reset the data register pointer
```

The right-hand joystick has a smaller range than the left.

Note that the servo calibration values in this .ino file will not be correct for your servos.